Q1. **Import the attached CSV files (Diamond.csv) and answer the following questions:**

1. Create 2 dataframes out of this dataframe – 1 with all numerical variables and other with all categorical variables.
2. Calculate the measure of central tendency of numerical variables using Pandas and statistics libraries and check if the calculated values are different between these 2 libraries.
3. Check the skewness of all numeric variables. Mention against each variable if its highly skewed/light skewed/ Moderately skwewed.
4. Use the different transformation techniques to convert skewed data found in previous question into normal distribution.
5. Create a user defined function in python to check the outliers using IQR method. Then pass all numeric variables in that function to check outliers.
6. Convert categorical variables into numerical variables using LabelEncoder technique.
7. Use both the feature scaling techniques (standardscaler/min max scaler) on all the variables.
8. Create the Histogram for all numeric variables and draw the KDE plot on that.
9. Check the correlation between all the numeric variables using HeatMap and try to draw some conclusion about the data.

**Ans:**

[**https://colab.research.google.com/drive/1GYlkPjgiTsLz1PTn\_ICvmd-7PeP2ECb\_?usp=sharing**](https://colab.research.google.com/drive/1GYlkPjgiTsLz1PTn_ICvmd-7PeP2ECb_?usp=sharing)

**Q2 Explain Gradient descent in detail. How changing the values of learning rate can impact the convergence in Gradient Descent.  
  
Ans:**

Of course. Here is a detailed explanation of Gradient Descent and the critical role of the learning rate.

### What is Gradient Descent? The Core Idea

At its heart, Gradient Descent is an optimization algorithm used to find the minimum value of a function.

In machine learning, this "function" is the cost function (or loss function), which measures how bad our model's predictions are compared to the actual data. The goal of training a model is to find the set of model parameters (e.g., the slope and intercept of a line) that result in the lowest possible cost.

The Analogy: Hiking Down a Mountain in the Fog

Imagine you are on a mountain in a thick fog and you want to get to the lowest point (the valley).

1. You can't see the whole map. The fog is so thick you can only see the ground right around your feet.
2. What's your strategy? You would look at the slope of the ground where you are standing and take a step in the steepest downhill direction.
3. Repeat. From your new position, you again look for the steepest downhill direction and take another step.
4. Keep going. You repeat this process until you reach a point where the ground is flat. You assume this is the bottom of the valley.

In this analogy:

* The Mountain: The cost function's surface. The height at any point is the "cost" or "error."
* Your Position: The current values of your model's parameters (e.g., m and c for a line y = mx + c).
* The Direction of the Steepest Slope: This is the gradient.
* The Size of Your Step: This is the learning rate.
* The Bottom of the Valley: The point of minimum cost, where the model's parameters are optimal.

### The Technical Details: How It Works

Let's break down the mathematical components.

1. Cost Function (J(θ)): This is the function we want to minimize. It takes the model parameters (represented by the vector θ) as input and outputs a single number representing the model's error. A common example is Mean Squared Error (MSE).
2. Gradient (∇J(θ)): The gradient is a vector of partial derivatives. In simple terms, it points in the direction of the steepest *ascent* (uphill) on the cost function's surface.
   * For each parameter in θ, the partial derivative tells us how much the cost function J would change if we made a tiny change to that one parameter.
   * Since the gradient points *uphill*, we must move in the opposite direction of the gradient to go downhill.
3. Learning Rate (α - alpha): This is a small positive number (e.g., 0.01, 0.1) that controls how big of a step we take in the downhill direction. It's a critical hyperparameter that you, the data scientist, must set.
4. The Update Rule: This is the core formula of Gradient Descent. In each iteration, we update every parameter in our model using this rule:

θ\_new = θ\_old - α \* ∇J(θ)

Let's break it down:

* + θ\_new: The new, updated value for a parameter.
  + θ\_old: The current value of the parameter.
  + -: The minus sign is crucial. It means we move in the *opposite* direction of the gradient (i.e., downhill).
  + α: The learning rate, which scales the size of our step.
  + ∇J(θ): The gradient, which tells us the direction of the step.

### The Algorithm in Steps

1. Initialize: Start with random values for your model parameters (θ).
2. Loop/Iterate: Repeat the following until the model converges:  
   a. Calculate the Gradient: Compute the gradient of the cost function with respect to the current parameters.  
   b. Update Parameters: Apply the update rule to get the new, slightly better parameters.
3. Stop: Convergence occurs when the steps become infinitesimally small or the change in cost is negligible, meaning you've reached a minimum.

**Excellent request. Connecting the optimization process to the final model's state (underfitting/overfitting) is a crucial step in deeply understanding machine learning.**

Here is a more technical modification of the explanation, framing the impact of the learning rate in terms of bias, variance, and model generalization.

### How the Learning Rate (α) Influences Generalization and Leads to Underfitting or Overfitting

The choice of the learning rate (α) is a critical optimization decision that directly impacts the training dynamics. While its primary role is to guide the model's parameters toward a minimum of the cost function, its value has profound consequences on the final model's ability to generalize to unseen data, often manifesting as underfitting (high bias) or overfitting (high variance).

The ultimate goal is not just to minimize the cost on the *training data*, but to find a set of parameters (θ) that represent a general solution and perform well on new, unseen test data.

#### 1. How a Poor Learning Rate Leads to Underfitting (High Bias)

An underfit model is too simple and fails to capture the underlying patterns in the data. It has high bias because its predictions are systematically wrong. This state can be a direct result of improper learning rate selection.

Case A: Learning Rate is Too Small (e.g., α = 0.0001)

* Optimization Behavior: With a tiny learning rate, the parameter updates (α \* ∇J(θ)) are minuscule. The optimizer makes extremely slow progress down the cost function's surface.
* Consequence on Model State:
  + If training is conducted for a fixed or insufficient number of epochs, the training process will terminate prematurely. The model's parameters will be "stuck" far from the optimal values that represent the lowest cost. The algorithm simply hasn't had enough time to learn. The resulting model is underfit. It has not converged to a meaningful solution, fails to model the complexity of the training data, and consequently performs poorly on both the training and test sets.
  + If model was trained with this then, it will **overfit** the data and instead of generalization it will learn it.

Case B: Learning Rate is Too Large (Causing Divergence)

* Optimization Behavior: If the learning rate is excessively large, the parameter update step (α \* ∇J(θ)) is so massive that it consistently overshoots the minimum. The cost function *increases* with each iteration, leading to divergence. The parameter values often explode towards infinity.
* Consequence on Model State: A diverged model has learned nothing. Its parameters are effectively noise.
* Result: This is the most extreme form of underfitting. The model is completely non-functional and has a higher error than a simple baseline model (e.g., one that always predicts the mean).

#### 2. How a Poor Learning Rate Can Contribute to Overfitting (High Variance)

An overfit model is too complex and has learned the noise in the training data, not just the signal. It has high variance because its performance is highly sensitive to the specific training data it saw.

Case A: Learning Rate is Too Large (but not Diverging)

* Optimization Behavior: A learning rate that is large, but not large enough to cause divergence, will often lead to an unstable and oscillating convergence path. The optimizer bounces around erratically in the cost function landscape.
* Consequence on Model State: This erratic behavior makes the optimizer more likely to settle into a "sharp minimum" of the cost function. A sharp minimum is a narrow, steep valley that corresponds to a highly specific set of parameter values that work perfectly for the training data's noise but are not robust.
* Result: A model that has converged to a sharp minimum is often overfit. It exhibits excellent performance on the training data but fails to generalize to the test data. A tiny change in the input data (as found in the test set) pushes the model out of the sharp valley, causing a large increase in error.

#### 3. The "Just Right" Learning Rate: Achieving a Good Fit (Low Bias & Low Variance)

* Optimization Behavior: An optimal learning rate allows for a smooth convergence path. It's large enough to escape shallow local minima and make rapid initial progress, but small enough to avoid overshooting and to settle gracefully into the minimum.
* Consequence on Model State: This smooth descent is more likely to find a "flat" or "wide" minimum. A flat minimum is a broad, gentle valley in the cost function landscape. Models that converge to flat minima are more robust and generalize better.
* Result: A model in a flat minimum is less sensitive to small perturbations in its parameters or the input data. This robustness is the hallmark of a well-generalized model that has found a balance between bias and variance—a good fit. Techniques like learning rate decay (gradually decreasing α during training) are explicitly designed to encourage this behavior: take large steps initially, then smaller, more careful steps to find the center of a wide, stable minimum.

### Technical Summary

|  |  |  |
| --- | --- | --- |
| Learning Rate (α) Setting | Optimization Behavior | Consequence on Model State & Generalization |
| Too Small | Very slow convergence; insufficient training. | Underfitting (High Bias): Model never reaches optimal parameters. |
| Too Large (Diverging) | Cost function increases; parameters explode. | Extreme Underfitting (High Bias): Model is non-functional. |
| Too Large (Oscillating) | Unstable convergence; bounces around the minimum. | Contributes to Overfitting (High Variance): Tends to find sharp minima that don't generalize. |
| "Just Right" | Smooth, efficient convergence to a stable point. | Good Fit (Low Bias/Variance): More likely to find flat minima that represent robust, general solutions. |